Predicting general market trajectory via adaptive LMS algorithm

Sean Conrad M. Atangan
College of Engineering,
University of Illinois at Chicago,
1200 W Harrison St,
Chicago, IL,
United States

Eric Escobedo
College of Engineering,
University of Illinois at Chicago,
1200 W Harrison St,
Chicago, IL,
United States

***Abstract*—Background: The market is misunderstood as a largely unpredictable force that requires a deep understanding of economics and human psychology. We assert that while there is truth to this statement, it is still possible to predict certain components of how the stock market functions to supplement a more holistic picture of its behavior. Machine learning can be utilized to predict general trends in said behavior, and the identification of said trends can be used in conjunction with observations in the news to provide a more informed analysis of the possible futures of stock market behavior.**

Methods: In this study, we deployed the adaptive LMS algorithm to predict the following day's market close values for various stocks and cryptocurrencies.

Results: We found that historical quotes spanning a larger time frame yield a more accurate prediction for the following day. We also observe that larger filters reduce the accuracy of the prediction across all tested assets. Lastly, we found that the prediction for a cryptocurrency asset was significantly less accurate than all stock market assets tested in this experiment, suggesting the possibility of less effective predictions from the algorithm for more volatile markets, though this is not vigorously tested for in this experiment.

Conclusion: We conclude that the adaptive LMS algorithm is indeed an effective method for predicting the general trend of an asset's price movement, and that it should be considered when developing more effective methods of price prediction.

I. Introduction

This study aims to identify useful parameter configurations when determining the general trajectory of prices in the stock and cryptocurrency market. We consider historical quotes of varying time lengths, as well as employ the adaptive LMS algorithm for predicting the closing price of each stock for the following day. This method is not enough to accurately predict the future prices of stocks and cryptocurrencies in a reliable manner but should be used in conjunction with other methods if practical results are to be obtained.

II. Method Description

A. Selecting a stock or cryptocurrency

First, the user chooses a stock or cryptocurrency to analyze. In this study, we provide three different options to choose from. These options are either Google, Apple, Dogecoin, or Amazon. The user is then given the option to choose a one month, three-month, or six-month version of the historical quote for that particular asset. These were obtained from the NASDAQ stock data base [3].

B. Implementing the LMS algorithm

Next, the user is then prompted to enter the filter size to be used. Once the stock data and filter size are chosen, an array is filled with data from the stock excel sheet. The filter determines how many data entries will be used at a given time to predict a subsequent data entry. For example, if a filter of size two is chosen, then the first and second data entries are used to predict the value of the third data entry.

To predict the subsequent value of the data entries that fit inside the filter, the filter values are multiplied by a weight vector. This weight vector is designed to be a scalar that adjusts the values of data entries to predict the next data entry more accurately [1]. This weight vector is updated throughout the process so that it can become more accurate. The weights are initially set to one, and the size of the weight vector matches the size of the filter. The reason for this is that each index of the weight vector is multiplied against each corresponding index of the stock filter. For example, if we are currently working with the third and fourth indices of the stock data and our filter is of size two, then the first index of the weight vector is multiplied with the third index of the stock array, and the second index of the weight vector is multiplied by the fourth index of the stock array. Once all the weight values are multiplied by their corresponding stock array values, the summation of these products will be equal to the predicted value [2].

Next, we need to compare our predicted value to the actual stock value predicted by subtracting them. The difference will be used to update the weights so that they will be more accurate for the next set of data entries. The weights will be updated by subtracting a weight updater from them. To obtain the weight updater, we need two things: the current stock filter we are using and a scalar to multiply it by.

The scalar is a fraction, and its numerator is the difference of our predicted value and the actual stock value. The denominator is the sum of all the stock values squared that there were used to predict the current stock value in the filter. This scalar is then

multiplied to an array of stock values inside the current filter. This becomes the weight updater (1).

$$w(n+1)=w(n) + 1\ 2\ \mu[-\nabla J(n)] \quad (1)$$

Now that we have the weight updater, we subtract this from the current weights to obtain new weights that are more accurate to the stock dataset we are currently working with. This process is then repeated as our filter shifts through the entire stock dataset. As an example, if our filter size is initially three, we then predict the fourth data point by using the LMS algorithm against the first three indices. We then compare the predicted value with the actual value of the fourth index. Once we update our weights, we then shift the filter to work with the second, third, and fourth indices to predict the fifth index using the updated weights. This updates continuously until the filter has reached the end of the stock data.

III. Experimental Results

After the completion of this experiment, we observed that the LMS algorithm was effective in predicting the future market values for the following day. Upon testing the algorithm for four different assets with historical quotes spanning a total of one month's worth of close value's, we found that the prediction for Google was as high as 96.83%. Similar results were observed for the other stock market values of Apple and Amazon, landing a maximum of 97.07% and 97.03% respectively. We discovered that for assets that exhibit greater volatility, the algorithm was significantly less accurate; such was the case for the cryptocurrency Dogecoin.

A. Longer List of Historical Quotes

We tested the algorithm for differences in accuracy using varying lengths of historical quote time frames. Historical quotes ranged from April 28th, 2021, to October 29th, 2020.

It was also observed that an increase in the length of the span of the historical quotes was directly proportional to the increase of the algorithm's accuracy. Upon completing the experiment, we observed that the accuracy of the algorithm was 1.49% better when training the algorithm with three months' worth of historical data, as opposed to the initial one month. Testing the algorithm with six months' worth of historical quotes also yielded a maximum accuracy of 1.91% better than the same initial month. This trend remained consistent for the rest of the assets, including the only cryptocurrency tested in this experiment, yielding an increase of 3.36% for Dogecoin.

B. Increasing Filter Size

We tested the algorithm's accuracy at varying filter sizes to observe what effect it had on the algorithm's performance.

It was observed that increasing the filter size resulted in a decrease in the algorithm's accuracy. This trend was observed across all assets and was consistent throughout all time spans of historical quotes. Increasing the filter size from two to ten for Google yielded a loss of 10.89% accuracy. Similar losses were observed at longer time spans.

C. Figures and Tables

TABLE I. Google Historical Quotes

Filter Size	Google Prediction Accuracy		
	1 month	*3 months*	*6 months*
2	96.83%	98.32%	98.74%
3	94.99%	97.27%	97.73%
4	93.78%	96.76%	97.23%
5	92.46%	96.51%	96.86%
6	91.27%	96.38%	96.67%
7	90.00%	95.80%	96.28%
8	88.57%	95.60%	96.17%
9	87.24%	95.40%	96.01%
10	85.94%	95.10%	95.73%

TABLE II. Apple Historical Quotes

Filter Size	Apple Prediction Accuracy		
	1 month	*3 months*	*6 months*
2	97.07%	98.31%	98.54%
3	95.23%	97.11%	97.48%
4	94.48%	96.52%	96.81%
5	93.54%	96.07%	96.40%
6	92.88%	95.90%	96.12%
7	92.10%	95.40%	95.77%
8	91.71%	95.10%	95.57%
9	90.84%	94.60%	95.30%
10	90.24%	94.27%	95.14%

TABLE III. DOGECOIN HISTORICAL QUOTES

Filter Size	Dogecoin Prediction Accuracy		
	1 month	*3 months*	*6 months*
2	89.57%	90.10%	92.93%
3	83.88%	89.98%	91.28%
4	77.91%	86.56%	88.32%
5	73.43%	84.99%	86.38%
6	70.01%	82.94%	84.77%
7	66.82%	80.84%	82.82%
8	62.78%	79.78%	81.09%
9	60.63%	78.24%	79.11%
10	57.97%	76.60%	77.54%

TABLE IV. AMAZON HISTORICAL QUOTES

Filter Size	Amazon Prediction Accuracy		
	1 month	*3 months*	*6 months*
2	97.03%	98.39%	98.69%
3	95.32%	97.33%	97.79%
4	94.29%	96.65%	97.17%
5	93.55%	96.26%	96.87%
6	92.83%	95.87%	96.67%
7	92.09%	95.47%	96.51%
8	91.68%	95.22%	96.37%
9	91.32%	95.00%	96.28%
10	90.73%	94.76%	96.14%

IV. CONCLUSION

From the data that we have collected, we conclude that the adaptive LMS algorithm is an effective method for predicting future prices in the stock and cryptocurrency market. While this experiment does not account for random occurrences that may also affect the prices of assets in the market, such as real-world events directly related to an asset, the adaptive LMS algorithm still stands as an effective predictor for the general trajectory of an asset's price and should be considered in parallel with other methods as far as future price prediction is concerned. Possible extensions for this experiment include the analysis the general connotation of Twitter tweets that contain an asset's name, as well as incorporating market volume and relative strength index as layers in a convolutional neural network for the purpose of obtaining a more accurate result.

CONTRIBUTIONS

Eric Escobedo:

I created the python functions that uses the LMS algorithm on imported stock data. I broke up several components of the algorithm and made them all separate functions that could be called in one master function at the end. I made the program scalable so that it could work with any dataset, not matter how large. I converted formulas and equations used in our ECE 491 class into programmable functions that could work for a wide range of inputs. I created weight functions that could adapt as the program went along to accurately predict future values of the stock dataset.

Sean Conrad M. Atangan:

I joined all functions into a main loop to be utilized in a user-friendly manner through the console and built it such that the user can customize the parameters for specific testing purposes. I configured the program to loop indefinitely unless otherwise directed by the user's input. I also prepared the historical quote data for each asset and created 1-month, 3-month, and 6-month versions for each asset to better supplement our testing as well as quantify the effectiveness of varying datasets. I built the getAccuracy function as well as implemented average accuracy functionality for each historical quote to observe the performance of the algorithm under different parametric configurations. I performed all the testing and logged the results into this report.

REFERENCES

[1] M. Stridh Class Lecture, Topic: "Lecture 2- LMS Algorithm", College of Engineering, Lund University, Sweden, Apr., 2021

[2] A. Cetin ECE 491 Lecture, Topic: "LMS algorithms", College of Engineering, University of Illinois at Chicago, Apr 2021

[3] NASDAQ, 2021. Accessed on: April 1, 2021. [Online]. Available: https://www.nasdaq.com/

APPENDIX

```
import pandas as pd
import xlrd
import openpyxl

def fillStock(stockArray, stockData):
    i = 2
    #numSales = len(stockData)
    numSales = stockData.max_row
    print("number of sales: ", numSales)
    while i <= numSales:
        stockArray.append(stockData.cell(row = i, column = 2).value)
        i = i + 1
    stockArray.reverse()
    print(stockArray)
    return stockArray

def predict(stockArray, weights, index):
    i = 0
    total = 0
    while i < len(weights):
        total = total + (stockArray[index + i] * weights[i])
        i = i + 1
    print(total)
    return total
#predict(dataStock, weights, 1)

def errorRate(predicted, actual):
    return actual - predicted

def denominator(stockArray, size, index):
    i = 0
    total = 0
    while i < size:
        total = total + (stockArray[index + i]**2)
        i = i + 1
    return total

def weightDiff(stockArray, size, index, dom, error):
    i = 0
    weightSub = [0] * size
    fraction = (-1 * error) / dom
    while i < size:
        weightSub[i] = fraction * stockArray[index + i]
        i = i + 1
    return weightSub

def newWeight(old, new, size):
    i = 0
    finalWeight = [0] * size
    while i < size:
        finalWeight[i] = old[i] - new[i]
        i = i + 1
    return finalWeight
```

```
def getAccuracy(pred, actual):
    if pred > actual:
        return actual / pred
    else:
        return pred / actual

def master(stockArray, stockData, size):
    totalAccuracy = 0
    stockArray = fillStock(stockArray, stockData)
    #size refers to the number of data entries being fed into the LMS
    #algorithm at a single time
    weights = [1] * size
    print("Weights: ", weights)
    print("Stock:\n", stockArray)
    index = 0
    numSales = stockData.max_row
    while index <= (numSales - size):
        predicted = predict(stockArray, weights, index)
        error = errorRate(predicted, stockArray[index + size])
        dom = denominator(stockArray, size, index)
        weightSub = weightDiff(stockArray, size, index, dom, error)
        weights = newWeight(weights, weightSub, size)
        index = index + 1
        print(index, ": ", "Actual: ", stockArray[index], " Predicted: ", predicted)
        #accuracy = abs(1 - (abs(predicted - stockArray[index])/stockArray[index]))
        accuracy = getAccuracy(predicted, stockArray[index])
        totalAccuracy += accuracy
        print("Accuracy: ", accuracy , "\n")
    print("Average Accuracy: ", (totalAccuracy / index) * 100, "% \n")

print("Starting Program...")
while 1:
    userChoice = input("Enter stock or crypto name, 0 to exit\n")
    if userChoice == "Google" or userChoice == "Apple" or userChoice == "Doge" or userChoice == "Amazon":
        quoteLength = input("Enter quote length in months: 1m, 3m, or 6m\n")
        userFilter = input("Enter filter size\n")
        path = userChoice + quoteLength + ".xlsx"
        data = openpyxl.load_workbook(path)
        data_obj = data.active
        stock = data_obj.cell(row = 2, column = 2)
        dataStock = []
        fillStock(dataStock, data_obj)
        size = int(userFilter) #filter size
        weights = [1] * size
        print("weights: ", weights[1])
        master(dataStock, data_obj, size)

    elif userChoice == "0":
        print("Ending program...")
        break
    else:
        print("Invalid entry... Try again")
```